(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
28 March 2002 (28.03.2002)

PCT

(10) International Publication Number
WO 02/25409 A2

(51) International Patent Classification[7]: G06F 1/00

(21) International Application Number: PCT/CA01/01344

(22) International Filing Date:
20 September 2001 (20.09.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

60/234,152	21 September 2000 (21.09.2000)	US
60/235,354	26 September 2000 (26.09.2000)	US
60/270,663	20 February 2001 (20.02.2001)	US

(71) Applicant *(for all designated States except US)*: RESEARCH IN MOTION LIMITED [CA/CA]; 295 Phillip Street, Waterloo, Ontario N2L 3W8 (CA).

(72) Inventors; and

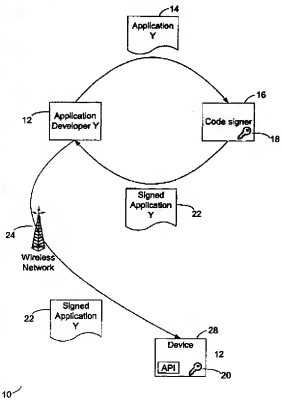
(75) Inventors/Applicants *(for US only)*: YACH, David, P. [CA/CA]; 254 Castlefield Avenue, Waterloo, Ontario N2K 2N1 (CA). BROWN, Michael, S. [CA/CA]; 7 Danube Street, Heidelberg, Ontario N0B 1Y0 (CA). LITTLE, Herbert, A. [CA/CA]; 504 Old Oak Place, Waterloo, Ontario N2T 2V8 (CA).

(74) Agent: PATHIYAL, Krishna, K.; Research In Motion Limited, 295 Phillip Street, Waterloo, Ontario N2L 3W8 (CA).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK,

[Continued on next page]

(54) Title: CODE SIGNING SYSTEM AND METHOD



(57) Abstract: A code signing system and method is provided. The code signing system operates in conjunction with a signed software application having a digital signature and includes an application platform, an application programming interface (API), and a virtual machine. The API is configured to link the software application with the application platform. The virtual machine verifies the authenticity of the digital signature in order to control access to the API by the software application.

WO 02/25409 A2

SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

Code Signing System And Method

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims priority from and is related to the following prior applications: "Code Signing System And Method," United States Provisional Application No. 60/234,152, filed September 21, 2000; "Code Signing System And Method," United States Provisional Application No. 60/235,354, filed September 26, 2000; and "Code Signing System And Method," United States Provisional Application No. 60/270,663, filed February 20, 2001.

BACKGROUND

1. FIELD OF THE INVENTION

This invention relates generally to the field of security protocols for software applications. More particularly, the invention provides a code signing system and method that is particularly well suited for Java™ applications for mobile communication devices, such as Personal Digital Assistants, cellular telephones, and wireless two-way communication devices (collectively referred to hereinafter as "mobile devices" or simply "devices").

2. DESCRIPTION OF THE RELATED ART

Security protocols involving software code signing schemes are known. Typically, such security protocols are used to ensure the reliability of software applications that are downloaded from the Internet. In a typical software code signing scheme, a digital signature is attached to a software application that identifies the software developer. Once the software is downloaded by a user, the user typically must use his or her judgment to determine whether or not the software

application is reliable, based solely on his or her knowledge of the software developer's reputation. This type of code signing scheme does not ensure that a software application written by a third party for a mobile device will properly interact with the device's native applications and other resources. Because typical code signing protocols are not secure and rely solely on the judgment of the user, there is a serious risk that destructive, "Trojan horse" type software applications may be downloaded and installed onto a mobile device.

There also remains a need for network operators to have a system and method to maintain control over which software applications are activated on mobile devices.

There remains a further need in 2.5G and 3G networks where corporate clients or network operators would like to control the types of software on the devices issued to its employees.

SUMMARY

A code signing system and method is provided. The code signing system operates in conjunction with a software application having a digital signature and includes an application platform, an application programming interface (API), and a virtual machine. The API is configured to link the software application with the application platform. The virtual machine verifies the authenticity of the digital signature in order to control access to the API by the software application.

A code signing system for operation in conjunction with a software application having a digital signature, according to another embodiment of the invention comprises an application platform, a plurality of APIs, each configured to link the software application with a resource on

the application platform, and a virtual machine that verifies the authenticity of the digital signature in order to control access to the API by the software application, wherein the virtual machine verifies the authenticity of the digital signature in order to control access to the plurality of APIs by the software application.

According to a further embodiment of the invention, a method of controlling access to sensitive application programming interfaces on a mobile device comprises the steps of loading a software application on the mobile device that requires access to a sensitive API, determining whether or not the software application includes a digital signature associated with the sensitive API, and if the software application does not include a digital signature associated with the sensitive API, then denying the software application access to the sensitive API.

In another embodiment of the invention, a method of controlling access to an application programming interface (API) on a mobile device by a software application created by a software developer comprises the steps of receiving the software application from the software developer, reviewing the software application to determine if it may access the API, if the software application may access the API, then appending a digital signature to the software application, verifying the authenticity of a digital signature appended to a software application, and providing access to the API to software applications for which the appended digital signature is authentic.

A method of restricting access to a sensitive API on a mobile device, according to a further embodiment of the invention, comprises the steps of registering one or more software developers that are trusted to design software applications which access the sensitive API, receiving a hash of a software application, determining if the software application was designed by one of the registered software developers, and if the software application was designed by one

of the registered software developers, then generating a digital signature using the hash of the software application, wherein the digital signature may be appended to the software application, and the mobile device verifies the authenticity of the digital signature in order to control access to the sensitive API by the software application.

In a still further embodiment, a method of restricting access to application programming interfaces on a mobile device comprises the steps of loading a software application on the mobile device that requires access to one or more API, determining whether or not the software application includes a digital signature associated with the mobile device, and if the software application does not include a digital signature associated with the mobile device, then denying the software application access to the one or more APIs.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a diagram illustrating a code signing protocol according to one embodiment of the invention;

Fig. 2 is a flow diagram of the code signing protocol described above with reference to Fig. 1;

Fig. 3 is a block diagram of a code signing system on a mobile device;

Fig. 3A is a block diagram of a code signing system on a plurality of mobile devices;

Fig. 4 is a flow diagram illustrating the operation of the code signing system described above with reference to Fig. 3 and Fig. 3A;

Fig. 5 is a flow diagram illustrating the management of the code signing authorities described with reference to Fig. 3A; and

Fig. 6 is a block diagram of a mobile communication device in which a code signing system and method may be implemented.

DETAILED DESCRIPTION

Referring now to the drawing figures, Fig. 1 is a diagram illustrating a code signing protocol according to one embodiment of the invention. An application developer 12 creates a software application 14 (application Y) for a mobile device that requires access to one or more sensitive APIs on the mobile device. The software application Y 14 may, for example, be a Java application that operates on a Java virtual machine installed on the mobile device. An API enables the software application Y to interface with an application platform that may include, for example, resources such as the device hardware, operating system and core software and data models. In order to make function calls to or otherwise interact with such device resources, a software application Y must access one or more APIs. APIs can thereby effectively "bridge" a software application and associated device resources. In this description and the appended claims, references to API access should be interpreted to include access of an API in such a way as to allow a software application Y to interact with one or more corresponding device resources. Providing access to any API therefore allows a software application Y to interact with associated device resources, whereas denying access to an API prevents the software application Y from interacting with the associated resources. For example, a database API may communicate with a device file or data storage system, and access to the database API would provide for interaction between a software application Y and the file or data storage system. A user interface (UI) API would communicate with controllers and/or control software for such device components as a

screen, a keyboard, and any other device components that provide output to a user or accept input from a user. In a mobile device, a radio API may also be provided as an interface to wireless communication resources such as a transmitter and receiver. Similarly, a cryptographic API may be provided to interact with a crypto module which implements crypto algorithms on a device. These are merely illustrative examples of APIs that may be provided on a device. A device may include any of these example APIs, or different APIs instead of or in addition to those described above.

Preferably, any API may be classified as sensitive by a mobile device manufacturer, or possibly by an API author, a wireless network operator, a device owner or operator, or some other entity that may be affected by a virus or malicious code in a device software application. For instance, a mobile device manufacturer may classify as sensitive those APIs that interface with cryptographic routines, wireless communication functions, or proprietary data models such as address book or calendar entries. To protect against unauthorized access to these sensitive APIs, the application developer 12 is required to obtain one or more digital signatures from the mobile device manufacturer or other entity that classified any APIs as sensitive, or from a code signing authority 16 acting on behalf of the manufacturer or other entity with an interest in protecting access to sensitive device APIs, and append the signature(s) to the software application Y 14.

In one embodiment, a digital signature is obtained for each sensitive API or library that includes a sensitive API to which the software application requires access. In some cases, multiple signatures are desirable. This would allow a service provider, company or network operator to restrict some or all software applications loaded or updated onto a particular set of

mobile devices. In this multiple-signature scenario, all APIs are restricted and locked until a "global" signature is verified for a software application. For example, a company may wish to prevent its employees from executing any software applications onto their devices without first obtaining permission from a corporate information technology (IT) or computer services department. All such corporate mobile devices may then be configured to require verification of at least a global signature before a software application can be executed. Access to sensitive device APIs and libraries, if any, could then be further restricted, dependent upon verification of respective corresponding digital signatures.

The binary executable representation of software application Y 14 may be independent of the particular type of mobile device or model of a mobile device. Software application Y 14 may for example be in a write-once-run-anywhere binary format such as is the case with Java software applications. However, it may be desirable to have a digital signature for each mobile device type or model, or alternatively for each mobile device platform or manufacturer. Therefore, software application Y 14 may be submitted to several code signing authorities if software application Y 14 targets several mobile devices.

Software application Y 14 is sent from the application developer 12 to the code signing authority 16. In the embodiment shown in Fig. 1, the code signing authority 16 reviews the software application Y 14, although as described in further detail below, it is contemplated that the code signing authority 16 may also or instead consider the identity of the application developer 12 to determine whether or not the software application Y 14 should be signed. The code signing authority 16 is preferably one or more representatives from the mobile device

manufacturer, the authors of any sensitive APIs, or possibly others that have knowledge of the operation of the sensitive APIs to which the software application needs access.

If the code signing authority 16 determines that software application Y 14 may access the sensitive API and therefore should be signed, then a signature (not shown) for the software application Y 14 is generated by the code signing authority 16 and appended to the software application Y 14. The signed software application Y 22, comprising the software application Y 14 and the digital signature, is then returned to the application developer 12. The digital signature is preferably a tag that is generated using a private signature key 18 maintained solely by the code signing authority 16. For example, according to one signature scheme, a hash of the software application Y 14 may be generated, using a hashing algorithm such as the Secure Hash Algorithm SHA1, and then used with the private signature key 18 to create the digital signature. In some signature schemes, the private signature key is used to encrypt a hash of information to be signed, such as software application Y 14, whereas in other schemes, the private key may be used in other ways to generate a signature from the information to be signed or a transformed version of the information.

The signed software application Y 22 may then be sent to a mobile device 28 or downloaded by the mobile device 28 over a wireless network 24. It should be understood, however, that a code signing protocol according to the present invention is not limited to software applications that are downloaded over a wireless network. For instance, in alternative embodiments, the signed software application Y 22 may be downloaded to a personal computer via a computer network and loaded to the mobile device through a serial link, or may be acquired from the application developer 12 in any other manner and loaded onto the mobile device. Once

the signed software application Y 22 is loaded on the mobile device 28, each digital signature is preferably verified with a public signature key 20 before the software application Y 14 is granted access to a sensitive API library. Although the signed software application Y 22 is loaded onto a device, it should be appreciated that the software application that may eventually be executed on the device is the software application Y 14. As described above, the signed software application Y 22 includes the software application Y 14 and one or more appended digital signatures (not shown). When the signatures are verified, the software application Y 14 can be executed on the device and access any APIs for which corresponding signatures have been verified.

The public signature key 20 corresponds to the private signature key 18 maintained by the code signing authority 16, and is preferably installed on the mobile device along with the sensitive API. However, the public key 10 may instead be obtained from a public key repository (not shown), using the device 28 or possibly a personal computer system, and installed on the device 28 as needed. According to one embodiment of a signature scheme, the mobile device 28 calculates a hash of the software application Y 14 in the signed software application Y 22, using the same hashing algorithm as the code signing authority 16, and uses the digital signature and the public signature key 20 to recover the hash calculated by the signing authority 16. The resultant locally calculated hash and the hash recovered from the digital signature are then compared, and if the hashes are the same, the signature is verified. The software application Y 14 can then be executed on the device 28 and access any sensitive APIs for which the corresponding signature(s) have been verified. As described above, the invention is in no way limited to this particular illustrative example signature scheme. Other signature schemes,

including further public key signature schemes, may also be used in conjunction with the code signing methods and systems described herein.

Fig. 2 is a flow diagram 30 of the code signing protocol described above with reference to Fig. 1. The protocol begins at step 32. At step 34, a software developer writes the software application Y for a mobile device that requires access to a sensitive API or library that exposes a sensitive API (API library A). As discussed above, some or all APIs on a mobile device may be classified as sensitive, thus requiring verification of a digital signature for access by any software application such as software application Y. In step 36, application Y is tested by the software developer, preferably using a device simulator in which the digital signature verification function has been disabled. In this manner, the software developer may debug the software application Y before the digital signature is acquired from the code signing authority. Once the software application Y has been written and debugged, it is forwarded to the code signing authority in step 38.

In steps 40 and 42, the code signing authority reviews the software application Y to determine whether or not it should be given access to the sensitive API, and either accepts or rejects the software application. The code signing authority may apply a number of criteria to determine whether or not to grant the software application access to the sensitive API including, for example, the size of the software application, the device resources accessed by the API, the perceived utility of the software application, the interaction with other software applications, the inclusion of a virus or other destructive code, and whether or not the developer has a contractual obligation or other business arrangement with the mobile device manufacturer. Further details of managing code signing authorities and developers are described below in reference to Fig. 5.

If the code signing authority accepts the software application Y, then a digital signature, and preferably a signature identification, are appended to the software application Y in step 46. As described above, the digital signature may be generated by using a hash of the software application Y and a private signature key 18. The signature identification is described below with reference to Figs. 3 and 4. Once the digital signature and signature identification are appended to the software application Y to generate a signed software application, the signed software application Y is returned to the software developer in step 48. The software developer may then license the signed software application Y to be loaded onto a mobile device (step 50). If the code signing authority rejects the software application Y, however, then a rejection notification is preferably sent to the software developer (step 44), and the software application Y will be unable to access any API(s) associated with the signature.

In an alternative embodiment, the software developer may provide the code signing authority with only a hash of the software application Y, or provide the software application Y in some type of abridged format. If the software application Y is a Java application, then the device independent binary *.class files may be used in the hashing operation, although device dependent files such as *.cod files used by the assignee of the present application may instead be used in hashing or other digital signature operations when software applications are intended for operation on particular devices or device types. By providing only a hash or abridged version of the software application Y, the software developer may have the software application Y signed without revealing proprietary code to the code signing authority. The hash of the software application Y, along with the private signature key 18, may then be used by the code signing authority to generate the digital signature. If an otherwise abridged version of the software

application Y is sent to the code signing authority, then the abridged version may similarly be used to generate the digital signature, provided that the abridging scheme or algorithm, like a hashing algorithm, generates different outputs for different inputs. This ensures that every software application will have a different abridged version and thus a different signature that can only be verified when appended to the particular corresponding software application from which the abridged version was generated. Because this embodiment does not enable the code signing authority to thoroughly review the software application for viruses or other destructive code, however, a registration process between the software developer and the code signing authority may also be required. For instance, the code signing authority may agree in advance to provide a trusted software developer access to a limited set of sensitive APIs.

In still another alternative embodiment, a software application Y may be submitted to more than one signing authority. Each signing authority may for example be responsible for signing software applications for particular sensitive APIs or APIs on a particular model of mobile device or set of mobile devices that supports the sensitive APIs required by a software application. A manufacturer, mobile communication network operator, service provider, or corporate client for example may thereby have signing authority over the use of sensitive APIs for their particular mobile device model(s), or the mobile devices operating on a particular network, subscribing to one or more particular services, or distributed to corporate employees. A signed software application may then include a software application and at least one appended digital signature appended from each of the signing authorities. Even though these signing authorities in this example would be generating a signature for the same software application,

different signing and signature verification schemes may be associated with the different signing authorities.

Fig. 3 is a block diagram of a code signing system 60 on a mobile device 62. The system 60 includes a virtual machine 64, a plurality of software applications 66-70, a plurality of API libraries 72-78, and an application platform 80. The application platform 80 preferably includes all of the resources on the mobile device 62 that may be accessed by the software applications 66-70. For instance, the application platform may include device hardware 82, the mobile device's operating system 84, or core software and data models 86. Each API library 72-78 preferably includes a plurality of APIs that interface with a resource available in the application platform. For instance, one API library might include all of the APIs that interface with a calendar program and calendar entry data models. Another API library might include all of the APIs that interface with the transmission circuitry and functions of the mobile device 62. Yet another API library might include all of the APIs capable of interfacing with lower-level services performed by the mobile device's operating system 84. In addition, the plurality of API libraries 72-78 may include both libraries that expose a sensitive API 74 and 78, such as an interface to a cryptographic function, and libraries 72 and 76, that may be accessed without exposing sensitive APIs. Similarly, the plurality of software applications 66-70 may include both signed software applications 66 and 70 that require access to one or more sensitive APIs, and unsigned software applications such as 68. The virtual machine 64 is preferably an object oriented run-time environment such as Sun Micro System's J2ME™ (Java 2 Platform, Micro Edition), which manages the execution of all of the software applications 66-70 operating on the mobile device 62, and links the software applications 66-70 to the various API libraries 72-78.

Software application Y 70 is an example of a signed software application. Each signed software application preferably includes an actual software application such as software application Y comprising for example software code that can be executed on the application platform 80, one or more signature identifications 94 and one or more corresponding digital signatures 96. Preferably each digital signature 96 and associated signature identification 94 in a signed software application 66 or 70 corresponds to a sensitive API library 74 or 78 to which the software application X or software application Y requires access. The sensitive API library 74 or 78 may include one or more sensitive APIs. In an alternative embodiment, the signed software applications 66 and 70 may include a digital signature 96 for each sensitive API within an API library 74 or 78. The signature identifications 94 may be unique integers or some other means of relating a digital signature 96 to a specific API library 74 or 78, API, application platform 80, or model of mobile device 62.

API library A 78 is an example of an API library that exposes a sensitive API. Each API library 74 and 78 including a sensitive API should preferably include a description string 88, a public signature key 20, and a signature identifier 92. The signature identifier 92 preferably corresponds to a signature identification 94 in a signed software application 66 or 70, and enables the virtual machine 64 to quickly match a digital signature 96 with an API library 74 or 78. The public signature key 20 corresponds to the private signature key 18 maintained by the code signing authority, and is used to verify the authenticity of a digital signature 96. The description string 88 may for example be a textual message that is displayed on the mobile device when a signed software application 66 or 70 is loaded, or alternatively when a software application X or Y attempts to access a sensitive API.

Operationally, when a signed software application 68-70, respectively including a software application X, Z, or Y, that requires access to a sensitive API library 74 or 78 is loaded onto a mobile device, the virtual machine 64 searches the signed for an appended digital signature 96 associated with the API library 74 or 78. Preferably, the appropriate digital signature 96 is located by the virtual machine 64 by matching the signature identifier 92 in the API library 74 or 78 with a signature identification 94 on the signed software application. If the signed software application includes the appropriate digital signature 96, then the virtual machine 64 verifies its authenticity using the public signature key 20. Then, once the appropriate digital signature 96 has been located and verified, the description string 88 is preferably displayed on the mobile device before the software application X or Y is executed and accesses the sensitive API. For instance, the description string 88 may display a message stating that "Application Y is attempting to access API Library A," and thereby provide the mobile device user with the final control to grant or deny access to the sensitive API.

Fig. 3A is a block diagram of a code signing system 61 on a plurality of mobile devices 62E, 62F and 62G. The system 61 includes a plurality of mobile devices each of which only three are illustrated, mobile devices 62E, 62F and 62G. Also shown is a signed software application 70, including a software application Y to which two digital signatures 96E and 96F with corresponding signature identifications 94E and 94F have been appended. In the example system 61, each pair composed of a digital signature and identification, 94E/96E and 94F/96F, corresponds to a model of mobile device 62, API library 78, or associated platform 80. If signature identifications 94E and 94F correspond to different models of mobile device 62, then when a signed software application 70 which includes a software application Y that requires

access to a sensitive API library 78 is loaded onto mobile device 62E, the virtual machine 64 searches the signed software application 70 for a digital signature 96E associated with the API library 78 by matching identifier 94E with signature identifier 92. Similarly, when a signed software application 70 including a software application Y that requires access to a sensitive API library 78 is loaded onto a mobile device 62F, the virtual machine 64 in device 62F searches the signed software application 70 for a digital signature 96F associated with the API library 78. However, when a software application Y in a signed software application 70 that requires access to a sensitive API library 78 is loaded onto a mobile device model for which the application developer has not obtained a digital signature, device 62G in the example of Fig. 3A, the virtual machine 64 in the device 64G does not find a digital signature appended to the software application Y and consequently, access to the API library 78 is denied on device 62G. It should be appreciated from the foregoing description that a software application such as software application Y may have multiple device-specific, library-specific, or API-specific signatures or some combination of such signatures appended thereto. Similarly, different signature verification requirements may be configured for the different devices. For example, device 62E may require verification of both a global signature, as well as additional signatures for any sensitive APIs to which a software application requires access in order for the software application to be executed, whereas device 62F may require verification of only a global signature and device 62G may require verification of signatures only for its sensitive APIs. It should also be apparent that a communication system may include devices (not shown) on which a software application Y received as part of a signed software application such as 70 may execute without any signature verification. Although a signed software application has one or

more signatures appended thereto, the software application Y might possibly be executed on some devices without first having any of its signature(s) verified. Signing of a software application preferably does not interfere with its execution on devices in which digital signature verification is not implemented.

Fig. 4 is a flow diagram 100 illustrating the operation of the code signing system described above with reference to Figs. 3 and 3A. In step 102, a software application is loaded onto a mobile device. Once the software application is loaded, the device, preferably using a virtual machine, determines whether or not the software application requires access to any API libraries that expose a sensitive API (step 104). If not, then the software application is linked with all of its required API libraries and executed (step 118). If the software application does require access to a sensitive API, however, then the virtual machine verifies that the software application includes a valid digital signature associated any sensitive APIs to which access is required, in steps 106-116.

In step 106, the virtual machine retrieves the public signature key 20 and signature identifier 92 from the sensitive API library. The signature identifier 92 is then used by the virtual machine in step 108 to determine whether or not the software application has an appended digital signature 96 with a corresponding signature identification 94. If not, then the software application has not been approved for access to the sensitive API by a code signing authority, and the software application is preferably prevented from being executed in step 116. In alternative embodiments, a software application without a proper digital signature 96 may be purged from the mobile device, or may be denied access to the API library exposing the sensitive API but executed to the extent possible without access to the API library. It is also contemplated

that a user may be prompted for an input when signature verification fails, thereby providing for user control of such subsequent operations as purging of the software application from the device.

If a digital signature 96 corresponding to the sensitive API library is appended to the software application and located by the virtual machine, then the virtual machine uses the public key 20 to verify the authenticity of the digital signature 96 in step 110. This step may be performed, for example, by using the signature verification scheme described above or other alternative signature schemes. If the digital signature 96 is not authentic, then the software application is preferably either not executed, purged, or restricted from accessing the sensitive API as described above with reference to step 116. If the digital signature is authentic, however, then the description string 88 is preferably displayed in step 112, warning the mobile device user that the software application requires access to a sensitive API, and possibly prompting the user for authorization to execute or load the software application (step 114). When more than one signature is to be verified for a software application, then the steps 104-110 are preferably repeated for each signature before the user is prompted in step 112. If the mobile device user in step 114 authorizes the software application, then it may be executed and linked to the sensitive API library in step 118.

Fig. 5 is a flow diagram 200 illustrating the management of the code signing authorities described with reference to Fig. 3A. At step 210, an application developer has developed a new software application which is intended to be executable one or more target device models or types. The target devices may include sets of devices from different manufacturers, sets of device models or types from the same manufacturer, or generally any sets of devices having

particular signature and verification requirements. The term "target device" refers to any such set of devices having a common signature requirement. For example, a set of devices requiring verification of a device-specific global signature for execution of all software applications may comprise a target device, and devices that require both a global signature and further signatures for sensitive APIs may be part of more than one target device set. The software application may be written in a device independent manner by using at least one known API, supported on at least one target device with an API library. Preferably, the developed software application is intended to be executable on several target devices, each of which has its own at least one API library.

At step 220, a code signing authority for one target device receives a target-signing request from the developer. The target signing request includes the software application or a hash of the software application, a developer identifier, as well as at least one target device identifier which identifies the target device for which a signature is being requested. At step 230, the signing authority consults a developer database 235 or other records to determine whether or not to trust developer 220. This determination can be made according to several criteria discussed above, such as whether or not the developer has a contractual obligation or has entered into some other type of business arrangement with a device manufacturer, network operator, service provider, or device manufacturer. If the developer is trusted, then the method proceeds at step 240. However, if the developer is not trusted, then the software application is rejected (250) and not signed by the signing authority. Assuming the developer was trusted, at step 240 the signing authority determines if it has the target private key corresponding to the submitted target identifier by consulting a private key store such as a target private key database 245. If the target private key is found, then a digital signature for the software application is generated at step 260

and the digital signature or a signed software application including the digital signature appended to the software application is returned to the developer at step 280. However, if the target private key is not found at step 240, then the software application is rejected at step 270 and no digital signature is generated for the software application.

Advantageously, if target signing authorities follow compatible embodiments of the method outlined in Fig. 5, a network of target signing authorities may be established in order to expediently manage code signing authorities and a developer community code signing process providing signed software applications for multiple targets with low likelihood of destructive code.

Should any destructive or otherwise problematic code be found in a software application or suspected because of behavior exhibited when a software application is executed on a device, then the registration or privileges of the corresponding application developer with any or all signing authorities may also be suspended or revoked, since the digital signature provides an audit trail through which the developer of a problematic software application may be identified. In such an event, devices may be informed of the revocation by being configured to periodically download signature revocation lists, for example. If software applications for which the corresponding digital signatures have been revoked are running on a device, the device may then halt execution of any such software application and possibly purge the software application from its local storage. If preferred, devices may also be configured to re-execute digital signature verifications, for instance periodically or when a new revocation list is downloaded.

Although a digital signature generated by a signing authority is dependent upon authentication of the application developer and confirmation that the application developer has

been properly registered, the digital signature is preferably generated from a hash or otherwise transformed version of the software application and is therefore application-specific. This contrasts with known code signing schemes, in which API access is granted to any software applications arriving from trusted application developers or authors. In the code signing systems and methods described herein, API access is granted on an application-by-application basis and thus can be more strictly controlled or regulated.

Fig. 6 is a block diagram of a mobile communication device in which a code signing system and method may be implemented. The mobile communication device 610 is preferably a two-way communication device having at least voice and data communication capabilities. The device preferably has the capability to communicate with other computer systems on the Internet. Depending on the functionality provided by the device, the device may be referred to as a data messaging device, a two-way pager, a cellular telephone with data messaging capabilities, a wireless Internet appliance or a data communication device (with or without telephony capabilities).

Where the device 610 is enabled for two-way communications, the device will incorporate a communication subsystem 611, including a receiver 612, a transmitter 614, and associated components such as one or more, preferably embedded or internal, antenna elements 616 and 618, local oscillators (LOs) 613, and a processing module such as a digital signal processor (DSP) 620. As will be apparent to those skilled in the field of communications, the particular design of the communication subsystem 611 will be dependent upon the communication network in which the device is intended to operate. For example, a device 610 destined for a North American market may include a communication subsystem 611 designed to

operate within the Mobitex™ mobile communication system or DataTAC™ mobile communication system, whereas a device 610 intended for use in Europe may incorporate a General Packet Radio Service (GPRS) communication subsystem 611.

Network access requirements will also vary depending upon the type of network 919. For example, in the Mobitex and DataTAC networks, mobile devices such as 610 are registered on the network using a unique identification number associated with each device. In GPRS networks however, network access is associated with a subscriber or user of a device 610. A GPRS device therefore requires a subscriber identity module (not shown), commonly referred to as a SIM card, in order to operate on a GPRS network. Without a SIM card, a GPRS device will not be fully functional. Local or non-network communication functions (if any) may be operable, but the device 610 will be unable to carry out any functions involving communications over network 619, other than any legally required operations such as "911" emergency calling.

When required network registration or activation procedures have been completed, a device 610 may send and receive communication signals over the network 619. Signals received by the antenna 616 through a communication network 619 are input to the receiver 612, which may perform such common receiver functions as signal amplification, frequency down conversion, filtering, channel selection and the like, and in the example system shown in Fig. 6, analog to digital conversion. Analog to digital conversion of a received signal allows more complex communication functions such as demodulation and decoding to be performed in the DSP 620. In a similar manner, signals to be transmitted are processed, including modulation and encoding for example, by the DSP 620 and input to the transmitter 614 for digital to analog

conversion, frequency up conversion, filtering, amplification and transmission over the communication network 619 via the antenna 618.

The DSP 620 not only processes communication signals, but also provides for receiver and transmitter control. For example, the gains applied to communication signals in the receiver 612 and transmitter 614 may be adaptively controlled through automatic gain control algorithms implemented in the DSP 620.

The device 610 preferably includes a microprocessor 638 which controls the overall operation of the device. Communication functions, including at least data and voice communications, are performed through the communication subsystem 611. The microprocessor 638 also interacts with further device subsystems or resources such as the display 622, flash memory 624, random access memory (RAM) 626, auxiliary input/output (I/O) subsystems 628, serial port 630, keyboard 632, speaker 634, microphone 636, a short-range communications subsystem 640 and any other device subsystems generally designated as 642. APIs, including sensitive APIs requiring verification of one or more corresponding digital signatures before access is granted, may be provided on the device 610 to interface between software applications and any of the resources shown in Fig. 6.

Some of the subsystems shown in Fig. 6 perform communication-related functions, whereas other subsystems may provide "resident" or on-device functions. Notably, some subsystems, such as keyboard 632 and display 622 for example, may be used for both communication-related functions, such as entering a text message for transmission over a communication network, and device-resident functions such as a calculator or task list.

Operating system software used by the microprocessor 638, and possibly APIs to be accessed by software applications, is preferably stored in a persistent store such as flash memory 624, which may instead be a read only memory (ROM) or similar storage element (not shown). Those skilled in the art will appreciate that the operating system, specific device software applications, or parts thereof, may be temporarily loaded into a volatile store such as RAM 626. It is contemplated that received and transmitted communication signals may also be stored to RAM 626.

The microprocessor 638, in addition to its operating system functions, preferably enables execution of software applications on the device. A predetermined set of applications which control basic device operations, including at least data and voice communication applications for example, will normally be installed on the device 610 during manufacture. A preferred application that may be loaded onto the device may be a personal information manager (PIM) application having the ability to organize and manage data items relating to the device user such as, but not limited to e-mail, calendar events, voice mails, appointments, and task items. Naturally, one or more memory stores would be available on the device to facilitate storage of PIM data items on the device. Such PIM application would preferably have the ability to send and receive data items, via the wireless network. In a preferred embodiment, the PIM data items are seamlessly integrated, synchronized and updated, via the wireless network, with the device user's corresponding data items stored or associated with a host computer system thereby creating a mirrored host computer on the mobile device with respect to the data items at least. This would be especially advantageous in the case where the host computer system is the mobile device user's office computer system. Further applications, including signed software

applications as described above, may also be loaded onto the device 610 through the network 619, an auxiliary I/O subsystem 628, serial port 630, short-range communications subsystem 640 or any other suitable subsystem 642. The device microprocessor 638 may then verify any digital signatures, possibly including both "global" device signatures and API-specific signatures, appended to such a software application before the software application can be executed by the microprocessor 638 and/or access any associated sensitive APIs. Such flexibility in application installation increases the functionality of the device and may provide enhanced on-device functions, communication-related functions, or both. For example, secure communication applications may enable electronic commerce functions and other such financial transactions to be performed using the device 610, through a crypto API and a crypto module which implements crypto algorithms on the device (not shown).

In a data communication mode, a received signal such as a text message or web page download will be processed by the communication subsystem 611 and input to the microprocessor 638, which will preferably further process the received signal for output to the display 622, or alternatively to an auxiliary I/O device 628. A user of device 610 may also compose data items such as email messages for example, using the keyboard 632, which is preferably a complete alphanumeric keyboard or telephone-type keypad, in conjunction with the display 622 and possibly an auxiliary I/O device 628. Such composed items may then be transmitted over a communication network through the communication subsystem 611.

For voice communications, overall operation of the device 610 is substantially similar, except that received signals would preferably be output to a speaker 634 and signals for transmission would be generated by a microphone 636. Alternative voice or audio I/O

subsystems such as a voice message recording subsystem may also be implemented on the device 610. Although voice or audio signal output is preferably accomplished primarily through the speaker 634, the display 622 may also be used to provide an indication of the identity of a calling party, the duration of a voice call, or other voice call related information for example.

The serial port 630 in Fig. 6 would normally be implemented in a personal digital assistant (PDA)-type communication device for which synchronization with a user's desktop computer (not shown) may be desirable, but is an optional device component. Such a port 630 would enable a user to set preferences through an external device or software application and would extend the capabilities of the device by providing for information or software downloads to the device 610 other than through a wireless communication network. The alternate download path may for example be used to load an encryption key onto the device through a direct and thus reliable and trusted connection to thereby enable secure device communication.

A short-range communications subsystem 640 is a further optional component which may provide for communication between the device 624 and different systems or devices, which need not necessarily be similar devices. For example, the subsystem 640 may include an infrared device and associated circuits and components or a Bluetooth™ communication module to provide for communication with similarly-enabled systems and devices.

The embodiments described herein are examples of structures, systems or methods having elements corresponding to the elements of the invention recited in the claims. This written description may enable those skilled in the art to make and use embodiments having alternative elements that likewise correspond to the elements of the invention recited in the claims. The intended scope of the invention thus includes other structures, systems or methods

that do not differ from the literal language of the claims, and further includes other structures, systems or methods with insubstantial differences from the literal language of the claims.

For example, when a software application is rejected at step 250 in the method shown in Fig. 5, the signing authority may request that the developer sign a contract or enter into a business relationship with a device manufacturer or other entity on whose behalf the signing authority acts. Similarly, if a software application is rejected at step 270, authority to sign the software application may be delegated to a different signing authority. The signing of a software application following delegation of signing of the software application to the different authority can proceed substantially as shown in Fig. 5, wherein the target signing authority that received the original request from the trusted developer at step 220 requests that the software application be signed by the different signing authority on behalf of the trusted developer from the target signing authority. Once a trust relationship has been established between code signing authorities, target private code signing keys could be shared between code signing authorities to improve performance of the method at step 240, or a device may be configured to validate digital signatures from either of the trusted signing authorities.

In addition, although described primarily in the context of software applications, code signing systems and methods according to the present invention may also be applied to other device-related components, including but in no way limited to, commands and associated command arguments, and libraries configured to interface with device resources. Such commands and libraries may be sent to mobile devices by device manufacturers, device owners, network operators, service providers, software application developers and the like. It would be desirable to control the execution of any command that may affect device operation, such as a

command to change a device identification code or wireless communication network address for example, by requiring verification of one or more digital signatures before a command can be executed on a device, in accordance with the code signing systems and methods described and claimed herein.

We claim:

1. A code signing system for operation in conjunction with a software application having a digital signature, comprising:

an application platform;

an application programming interface (API) configured to link the software application with the application platform; and

a virtual machine that verifies the authenticity of the digital signature in order to control access to the API by the software application.

2. The code signing system of claim 1, wherein the virtual machine denies the software application access to the API if the digital signature is not authentic.

3. The code signing system of claim 1, wherein the virtual machine purges the software application if the digital signature is not authentic.

4. The code signing system of claim 1, wherein the code signing system is installed on a mobile device.

5. The code signing system of claim 1, wherein the digital signature is generated by a code signing authority.

6. A code signing system for operation in conjunction with a software application having a digital signature, comprising:

an application platform;

a plurality of application programming interfaces (APIs), each configured to link the software application with a resource on the application platform; and

a virtual machine that verifies the authenticity of the digital signature in order to control access to the API by the software application,

wherein the virtual machine verifies the authenticity of the digital signature in order to control access to the plurality of APIs by the software application.

7. The code signing system of claim 6, wherein the plurality of APIs are included in an API library.

8. The code signing system of claim 6, wherein one or more of the plurality of APIs is classified as sensitive, and wherein the virtual machine uses the digital signature to control access to the sensitive APIs.

9. The code signing system of claim 8, for operation in conjunction with a plurality of software applications, wherein one or more of the plurality of software applications has a digital signature, and wherein the virtual machine verifies the authenticity of the digital signature of each of the one or more of the plurality of software applications in order to control access to the sensitive APIs by each of the plurality of software applications.

10. The code signing system of claim 6, wherein the resource on the application platform comprises a wireless communication system.

11. The code signing system of claim 6, wherein the resource on the application platform comprises a cryptographic module which implements cryptographic algorithms.

12. The code signing system of claim 6, wherein the resource on the application platform comprises a data store.

13. The code signing system of claim 6, wherein the resource on the application platform comprises a user interface (UI).

14. The code signing system of claim 1, further comprising:

a plurality of API libraries each including a plurality of APIs, wherein the virtual machine controls access to the plurality of API libraries by the software application.

15. The code signing system of claim 14, wherein one or more of the plurality of API libraries is classified as sensitive, and wherein the virtual machine uses the digital signature to control access to the sensitive API libraries by the software application.

16. The code signing system of claim 15, wherein the software application includes a unique digital signature for each sensitive API library.

17. The code signing system of claim 16, wherein:

the software application includes a signature identification for each unique digital signature;

each sensitive API library includes a signature identifier; and

the virtual machine compares the signature identification and the signature identifier to match the unique digital signatures with sensitive API libraries.

18. The code signing system of claim 1, wherein the digital signature is generated using a private signature key, and the virtual machine uses a public signature key to verify the authenticity of the digital signature.

19. The code signing system of claim 18, wherein:

the digital signature is generated by applying the private signature key to a hash of the software application; and

the virtual machine verifies the authenticity of the digital signature by generating a hash of the software application to obtain a generated hash, applying the public signature key to the digital signature to obtain a recovered hash, and comparing the generated hash with the recovered hash.

20. The code signing system of claim 1, wherein the API further comprises:

a description string that is displayed by the mobile device when the software application attempts to access the API.

21. The code signing system of claim 1, wherein the application platform comprises an operating system.

22. The code signing system of claim 1, wherein the application platform comprises one or more core functions of a mobile device.

23. The code signing system of claim 1, wherein the application platform comprises hardware on a mobile device.

24. The code signing system of claim 23, wherein the hardware comprises a subscriber identity module (SIM) card.

25. The code signing system of claim 1, wherein the software application is a Java application for a mobile device.

26. The code signing system of claim 1, wherein the API interfaces with a cryptographic routine on the application platform.

27. The code signing system of claim 1, wherein the API interfaces with a proprietary data model on the application platform.

28. The code signing system of claim 1, wherein the virtual machine is a Java virtual machine installed on a mobile device.

29. A method of controlling access to sensitive application programming interfaces on a mobile device, comprising the steps of:

loading a software application on the mobile device that requires access to a sensitive application programming interface (API);

determining whether or not the software application includes a digital signature associated with the sensitive API; and

if the software application does not include a digital signature associated with the sensitive API, then denying the software application access to the sensitive API.

30. The method of claim 29, comprising the additional step of:

if the software application does not include a digital signature associated with the sensitive API, then purging the software application from the mobile device.

31. The method of claim 29, wherein the digital signature is generated by a code signing authority.

32. The method of claim 29, comprising the additional steps of:

if the software application includes a digital signature associated with the sensitive API, then verifying the authenticity of the digital signature; and

if the digital signature is not authentic, then denying the software application access to the sensitive API.

33. The method of claim 32, comprising the additional step of:

if the digital signature is not authentic, then purging the software application from the mobile device.

34. The method of claim 32, wherein the digital signature is generated by applying a private signature key to a hash of the software application, and wherein the step of verifying the authenticity of the digital signature is performed by a method comprising the steps of:

storing a public signature key that corresponds to the private signature key on the mobile device;

generating a hash of the software application to obtain a generated hash;

applying the public signature key to the digital signature to obtain a recovered hash; and

comparing the generated hash with the recovered hash.

35. The method of claim 34, wherein the digital signature is generated by calculating a hash of the software application and applying the private signature key.

36. The method of claim 29, comprising the additional step of:

displaying a description string that notifies a user of the mobile device that the software application requires access to the sensitive API.

37. The method of claim 36, comprising the additional step of:

receiving a command from the user granting or denying the software application access to the sensitive API.

38. A method of controlling access to an application programming interface (API) on a mobile device by a software application created by a software developer, comprising the steps of:

receiving the software application from the software developer;

reviewing the software application to determine if it may access the API;

if the software application may access the API, then appending a digital signature to the software application;

verifying the authenticity of a digital signature appended to a software application; and

providing access to the API to software applications for which the appended digital signature is authentic.

39. The method of claim 38, wherein the step of reviewing the software application is performed by a code signing authority.

40. The method of claim 38, wherein the step of appending the digital signature to the software application is performed by a method comprising the steps of:

calculating a hash of the software application; and

applying a signature key to the hash of the software application to generate the digital signature.

41. The method of claim 40, wherein the hash of the software application is calculated using the Secure Hash Algorithm (SHA1).

42. The method of claim 40, wherein the step of verifying the authenticity of a digital signature comprises the steps of:

providing a corresponding signature key on the mobile device;

calculating the hash of the software application on the mobile device to obtain a calculated hash;

applying the corresponding signature key to the digital signature to obtain a recovered hash; and

determining if the digital signature is authentic by comparing the calculated hash with the recovered hash.

43. The method of claim 42, comprising the further step of, if the digital signature is not authentic, then denying the software application access to the API.

44. The method of claim 42, wherein the signature key is a private signature key and the corresponding signature key is a public signature key.

45. A method of controlling access to a sensitive application programming interface (API) on a mobile device, comprising the steps of:

registering one or more software developers that are trusted to design software applications which access the sensitive API;

receiving a hash of a software application;

determining if the software application was designed by one of the registered software developers; and

if the software application was designed by one of the registered software developers, then generating a digital signature using the hash of the software application, wherein

the digital signature may be appended to the software application; and

the mobile device verifies the authenticity of the digital signature in order to control access to the sensitive API by the software application.

46. The method of claim 45, wherein the step of generating the digital signature is performed by a code signing authority.

47. The method of claim 45, wherein the step of generating the digital signature is performed by applying a signature key to the hash of the software application.

48. The method of claim 47, wherein the mobile device verifies the authenticity of the digital signature by performing the additional steps of:

providing a corresponding signature key on the mobile device;

calculating the hash of the software application on the mobile device to obtain a calculated hash;

applying the corresponding signature key to the digital signature to obtain a recovered hash;

determining if the digital signature is authentic by comparing the calculated hash with the recovered hash; and

if the digital signature is not authentic, then denying the software application access to the sensitive API.

49. A method of restricting access to application programming interfaces on a mobile device, comprising the steps of:

loading a software application on the mobile device that requires access to one or more application programming interface (API);

determining whether or not the software application includes an authentic digital signature associated with the mobile device; and

if the software application does not include an authentic digital signature associated with the mobile device, then denying the software application access to the one or more APIs.

50. The method of claim 49, comprising the additional step of:

if the software application does not include an authentic digital signature associated with the mobile device, then purging the software application from the mobile device.

51. The method of claim 49, wherein:

the software application includes a plurality of digital signatures; and

the plurality of digital signatures includes digital signatures respectively associated with different types of mobile devices.

52. The method of claim 51, wherein each of the plurality of digital signatures is generated by a respective corresponding code signing authority.

53. The method of claim 49, wherein the step of determining whether or not the software application includes an authentic digital signature associated with the mobile device comprises the additional steps of:

determining if the software application includes a digital signature associated with the mobile device; and

if so, then verifying the authenticity of the digital signature.

54. The method of claim 53, wherein the one or more APIs includes one or more APIs classified as sensitive, and the method further comprises the steps of, for each sensitive API:

determining whether or not the software application includes an authentic digital signature associated with the sensitive API; and

if the software application does not include an authentic digital signature associated with the sensitive API, then denying the software application access to the sensitive API.

55. The method of claim 52, wherein each of the plurality of digital signatures is generated by its corresponding code signing authority by applying a respective private signature key associated with the code signing authority to a hash of the software application.

56. The method of claim 55, wherein the step of determining whether or not the software application includes an authentic digital signature associated with the mobile device comprises the steps of:

determining if the software application includes a digital signature associated with the mobile device; and

if so, then verifying the authenticity of the digital signature,

wherein the step of verifying the authenticity of the digital signature is performed by a method comprising the steps of:

storing a public signature key on a mobile device that corresponds to the private signature key associated with the code signing authority which generates the signature associated with the mobile device;

generating a hash of the software application to obtain a generated hash;

applying the public signature key to the digital signature to obtain a recovered hash; and

comparing the generated hash with the recovered hash.

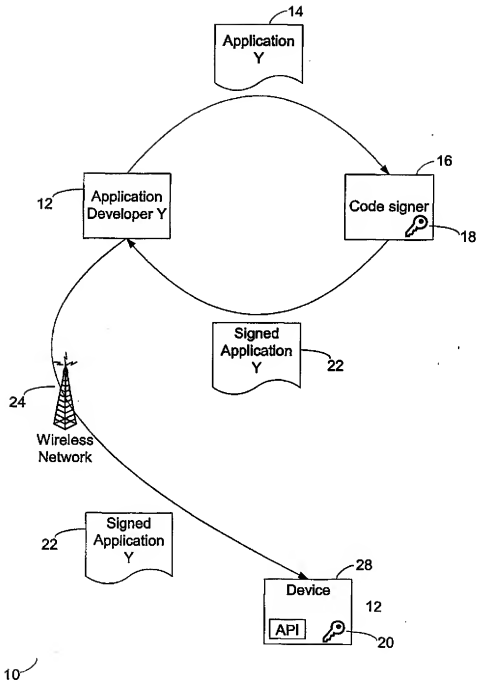
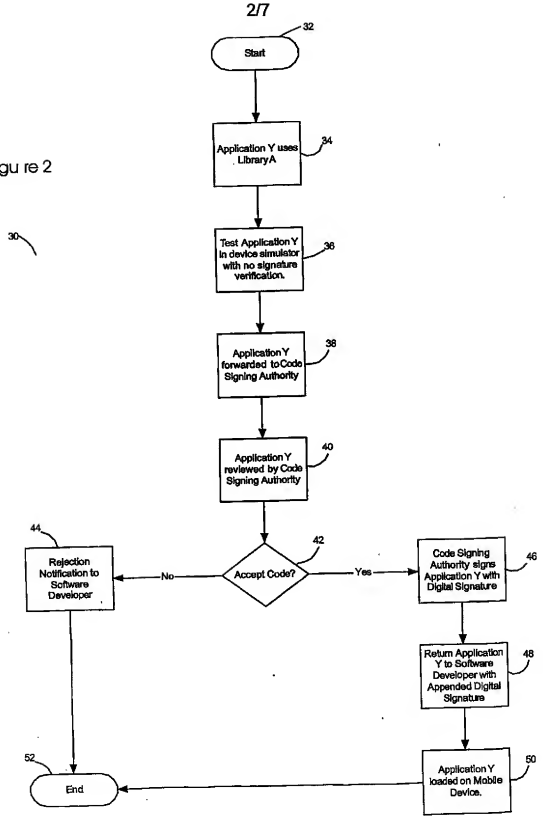

14
Application Y
16
12
Application Developer Y
Code signer
18
Signed Application Y
22
24
Wireless Network
22
Signed Application Y
28
Device
12
API
20
10

Figure 1

Figu re 2


30
32
Start
Application Y uses Library A
34
Test Application Y in device simulator with no signature verification.
36
Application Y forwarded to Code Signing Authority
38
Application Y reviewed by Code Signing Authority
40
Accept Code?
42
No
Yes
Rejection Notification to Software Developer
44
Code Signing Authority signs Application Y with Digital Signature
46
Return Application Y to Software Developer with Appended Digital Signature
48
Application Y loaded on Mobile Device.
50
End
52

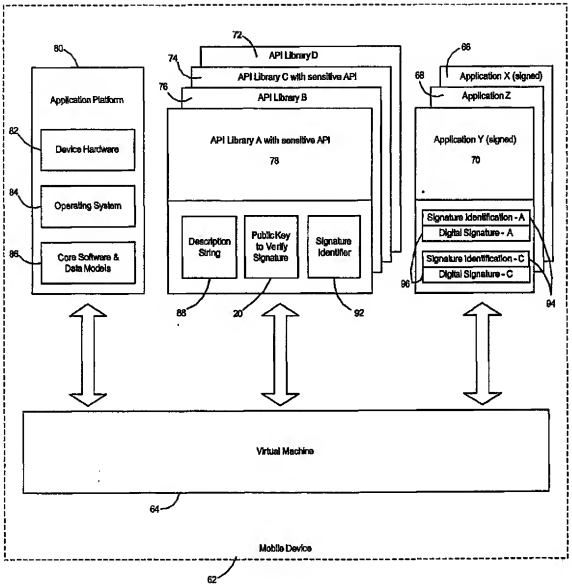

80
Application Platform
82
Device Hardware
84
Operating System
86
Core Software & Data Models
72
API Library D
74
API Library C with sensitive API
76
API Library B
API Library A with sensitive API
78
Description String
Public Key to Verify Signature
Signature Identifier
88
20
92
66
Application X (signed)
68
Application Z
Application Y (signed)
70
Signature Identification - A
Digital Signature - A
Signature Identification - C
Digital Signature - C
96
94
Virtual Machine
64
Mobile Device
62
60

Figure 3

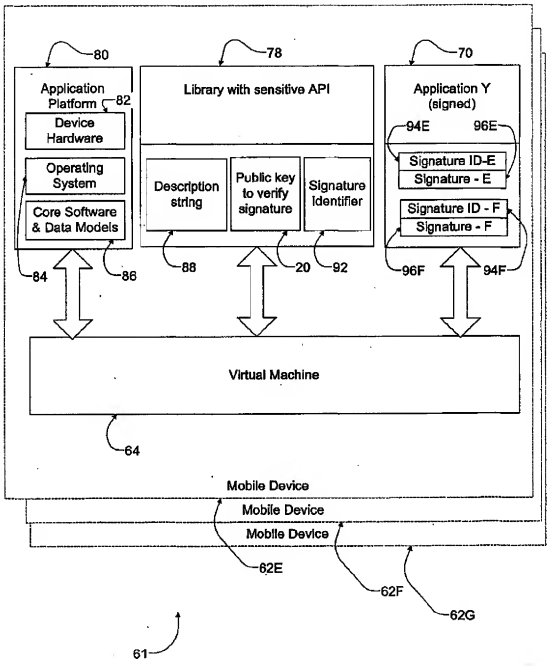
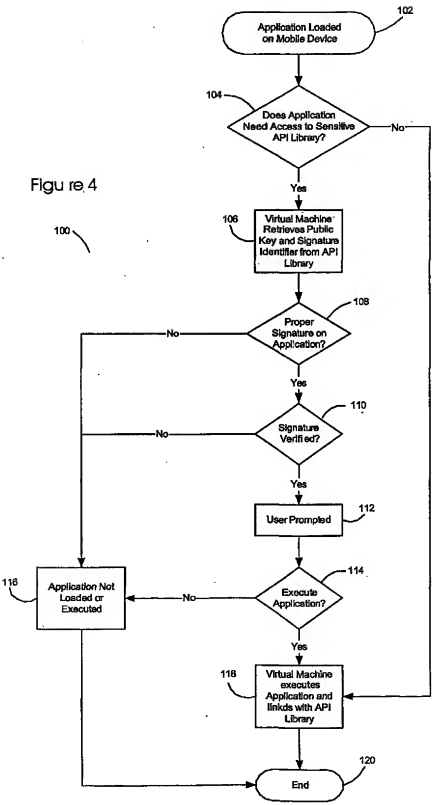

80
78
70
Application Platform
82
Device Hardware
Operating System
Core Software & Data Models
Library with sensitive API
Description string
Public key to verify signature
Signature Identifier
Application Y (signed)
94E
96E
Signature ID-E
Signature - E
Signature ID - F
Signature - F
84
86
88
20
92
96F
94F
Virtual Machine
64
Mobile Device
Mobile Device
Mobile Device
62E
62F
62G
61

Figure 3A

Figure 4


100
102
Application Loaded on Mobile Device
104
Does Application Need Access to Sensitive API Library?
No
Yes
106
Virtual Machine Retrieves Public Key and Signature Identifier from API Library
108
Proper Signature on Application?
No
Yes
110
Signature Verified?
No
Yes
112
User Prompted
114
Execute Application?
No
Yes
116
Application Not Loaded or Executed
118
Virtual Machine executes Application and linkds with API Library
120
End

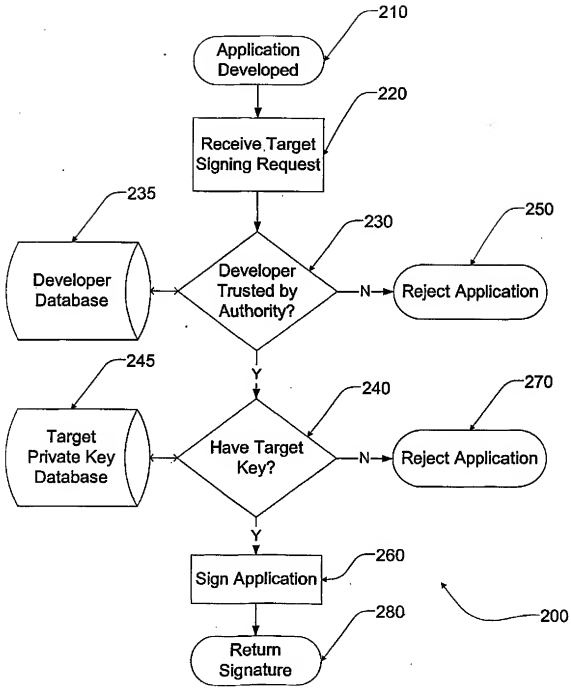

210
Application Developed
220
Receive Target Signing Request
235
Developer Database
230
Developer Trusted by Authority?
N
250
Reject Application
Y
245
Target Private Key Database
240
Have Target Key?
N
270
Reject Application
Y
260
Sign Application
280
Return Signature
200

Figure 5

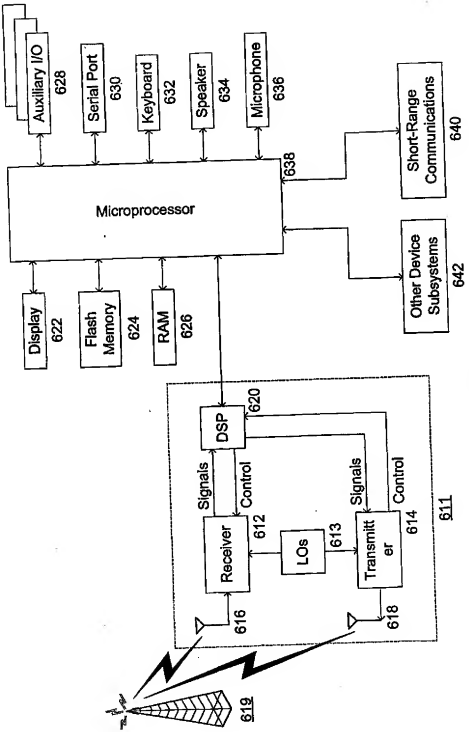

Auxiliary I/O
628
Serial Port
630
Keyboard
632
Speaker
634
Microphone
636
638
Microprocessor
Short-Range Communications
640
Other Device Subsystems
642
Display
622
Flash Memory
624
RAM
626
610
DSP
620
Signals
Control
Receiver
612
LOs
613
Signals
Control
Transmitter
614
611
616
618
619

Figure 6